05

Tratando el error 400

Transcripción

[00:00] Ya aprendimos cómo tratar un error y retornar un 404 en lugar de un 500 internal server error. Pero como les comenté, en el caso de registrar esto puede ser un poco diferente. ¿Por qué? Vamos a ver primero qué es lo que pasa, cómo es que Spring automáticamente mapea este error si yo lo que hago es quitarle estos tres parámetros a mi payload cuando yo intento registrar un médico, un recurso médico.

[00:29] Voy a enviarlo, e igual que en el caso anterior, me manda un timestamp, manda status 400, bad request. Hasta ahí estamos bien, porque si este es un bad request, es un request incorrecto, el payload es incorrecto. Entonces con el status code yo no tengo ningún problema, está perfecto. Pero el problema para mí comienza en esta parte de aquí.

[00:53] El array, el arreglo de errores. ¿Por qué? Si se dan cuenta, no es muy amigable y no es tan fácil de entender qué es lo que ha sucedido mal aquí, para nuestro cliente, nuestra aplicación cliente, sea móvil, sea web, en realidad es difícil entender qué es lo que ha pasado aquí. Por ejemplo aquí vemos que recién aquí “defaultMessage”: “no debe estar vacío”.

[01:18] ¿Qué es lo que no debe estar vacío? Documento. Y en el campo “documento” que está aquí. Entonces, vamos a intentar hacer esto un poco más amigable para nuestro cliente y porque ya estamos de acuerdo que el error que retorna es el correcto. Es un bad request, eso está fuera de discusión, pero el problema está a nivel del arreglo de errores que estamos retornando. ¿Cómo es, qué solucionamos eso?

[01:43] Venimos aquí. Y si yo les digo entonces que tenemos que tratar este tipo de excepción, ¿que se les viene a la mente a ustedes? Un exceptionHandler, entonces, vamos a repetir lo mismo que ya hemos hecho antes. Lo que voy a hacer, voy a copiar, voy a pegar y le voy a decir aquí tratarError400.

[02:08] Y aquí lo que le voy a decir que haga es un bad request. Y está bien, ¿dónde es la diferencia? En el tipo de excepción. Y si yo deseo saber el tipo de excepción que fue lanzado, voy a los logs y veo que el tipo de excepción lanzado es este de aquí: MethodArgumentNotValidException. Voy a ver si es verdad.

[02:32] Voy a venir aquí, voy a copiar aquí. Como ustedes pueden ver, toda la información que necesitan para tratamiento de excepciones la van a encontrar siempre en sus logs. Es muy importante que practiquen y aprendan a interpretar estos logs para saber qué es lo que está sucediendo en realidad, cuando tienen que debugear algún código o tratar algún tipo de error.

[02:53] Esto es solamente con la práctica, no se preocupen, poco a poco van a desarrollar esta habilidad. Bueno, vamos a guardar. Vamos a esperar a que se refresque nuestro servidor. Listo. Y vamos a ejecutarlo otra vez de la llamada. Vamos aquí y listo.

[03:16] Bad request. Hasta acá todo muy bien, retorna un bad request, pero ahora no estoy recibiendo ninguna información sobre cuál fue el parámetro que yo o no ingresé o ingresé mal, cierto, porque está siendo validado. Y esto sería un gran dolor de cabeza para nuestro cliente si solo le decimos tu payload está incorrecto pero no te digo en qué parte está incorrecto, entonces no puedes avanzar, ahí estaría bloqueado.

[03:44] ¿Cuál es el siguiente paso? Vamos a tratar de decirle a nuestro cliente en qué parte se ha equivocado. ¿Y esto que les dice? Que vamos a retornar algún tipo de cosa a nuestro cliente en esta parte aquí, vamos a retornar a algún body. Para esto quiero decir body. Necesito saber, con body no necesitamos el

.build porque automáticamente lo mapea, y aquí dentro debería haber alguna información para nuestro cliente.

[04:14] Segundo. Si la información cuando esta excepción era lanzada directamente y nos daba los detalles de los campos que habían fallado, eso me dice a mí que el lugar donde yo puedo encontrar esa información es en el mensaje de error de la excepción en sí. Quien contiene todos los errores encontrados es el mensaje dentro de esta excepción que está siendo lanzada por mi Bean Validator.

[04:48] Entonces, para yo obtener los datos de esta excepción aquí al método yo le puedo mandar un argumento del mismo tipo de la excepción que yo estoy lanzando aquí, esto tiene que ser exactamente igual. Con igual me refiero a la misma excepción, no puedo yo pretender hacer tratamiento de una excepción aquí y recibir como parámetro una excepción diferente. Eso no va a funcionar.

[05:18] ¿Entonces, qué más necesito? Yo necesito el mensaje de esta excepción. ¿La lista de errores de esta excepción, ¿cómo voy a hacer esto? Yo voy a crear aquí var errores y esto va a ser igual a e.getFieldError() Y errores, lo que yo voy a hacer es mandarlo aquí dentro del body.

[05:48] Voy a darle guardar. Voy a esperar que mi servidor refresque y vamos a ver qué es lo que vamos a obtener ahora. Vemos que ya refrescó, mandamos. Y nuevamente miren aquí, miren lo que tengo aquí. Tengo que en documento, en “datosRegistroMedico.documento” el error que fue lanzado, pero yo no solo tengo del documento, yo necesito los otros códigos de error.

[06:17] Entonces yo necesito acceder a la lista de errores que ha sido lanzada, entonces necesito ir a un nivel más arriba. ¿Dónde encuentro esa información? Aquí en getAllErrors. Punto y coma. Guardo ahora, reinicio mi servidor. Intentamos, y ahora sí, ya tenemos mi lista de errores que hay.

[06:50] Y si se dieron cuenta nuevamente tenemos el mismo error que no queríamos tener al inicio, una lista muy detallada y difícil de leer. Si yo quiero

intervenir aquí y personalizar el tipo de respuesta que quiero dar a mi cliente, ¿qué necesito? Un DTO. Entonces vengo aquí y como ese DTO va a ser usado solo aquí, a este nivel, yo lo voy a crear internamente aquí.

[07:16] Le voy a decir aquí a private record DatosErrorValidacion. Este DTO quiero que tenga un string llamado campo y el string llamado error. Es lo único que yo quiero que tenga mi DTO y nada más. Guardo aquí y por lo tanto lo que yo le voy a decir ahora es que a mis errores, errores en todo caso va a ser igual a punto.stream.map y voy a crear un (DatosErrorValidacion::new).toList().

[08:04] Porque yo quiero que de esta parte automáticamente sea mapeado a mis DatosErrorValidación y que esto se vuelva una lista. ¿Por qué no está compilando? Porque necesito aquí un constructor. Para eso le voy a hacer aquí un public DatosErrorValidacion.

[08:22] Y si se dieron cuenta, voy a dar un paso atrás en este momento porque si yo le digo que de mi excepción, el fieldError, ¿recuerdan el primer campo que vimos? Era un getFieldError. El tipo, el tipo de objeto es fieldError. Por lo tanto, el arreglo que yo tengo aquí en este momento es un arreglo de fieldErrors, entonces yo aquí voy a recibir un fieldError error.

[08:57] Y esto lo voy a mapear a simplemente error.getField() y error.getDefaultMessage(), acá está. Punto y coma al final. ¿Por qué? Porque yo ya sé que cada elemento de esta lista va a ser un elemento del campo fieldError. Eso lo descubrí porque primero llamé al método getFieldError para obtener solamente uno de los errores.

[09:38] Regresamos, le damos enviar. ¿Qué me dice? “Bad Requeest” “Validation failed for object”, no funcionó en este caso y me sigue retornando en mismo array de siempre. ¿Por qué? Porque nuestro método y nuestra validación falló en una parte aquí. Vamos a entender por qué.

[09:57] Me devuelve objetos según me dice aquí el error, voy a dar “Ctrl + Z”. GetAllErrors es una lista de objeto. Pero yo no necesito una lista de objetos, yo

necesito una lista de FieldError. Yo podría usar objeto, pero objeto, sabemos que es la clase padre y no tiene ningún campo digamos personalizado, porque es la más grande de todas.

[10:28] Entonces aquí yo necesito un arreglo de fieldErrors, que es este de aquí. Entonces, aquí es aquí getFieldErrors. Y aquí yo atrapo un FieldError. Le voy a dar "Ctrl + S" para guardar. Espero un poco que inicialice mi servidor nuevamente.

[10:52] Como pueden ver, hay muchos métodos aquí y a veces tenemos que ir buscando entre los distintos métodos que podemos tener aquí de esta excepción en particular para encontrar el que más nos convenga, de acuerdo a lo que queramos implementar.

[11:08] Le voy a dar send y vemos ahora finalmente “campo”: “documento”, que “no debe estar vacío”, “campo”: “nombre”, “no debe estar vacío” y el “campo”: “email”, que “no debe estar vacío”. Quiero probar si esto funciona, entonces lo que voy a hacer aquí es agregar, por ejemplo “Jimena Flores”, por ejemplo, con su nombre aquí.

[11:33] “Jimena.flores”, con un documento que no existe y con un teléfono ficticio también. Le voy a dar enviar. Y vemos que, en efecto, sigue funcionando mi método de crear, me retorna un 201 created con los datos correctos. Y si yo decido quitarle los parámetros obligatorios, me va a decir el “campo”: “email” “no debe estar vacío” y el “nombre” tampoco debe estar vacío.

[12:02] Y bien, ¿qué les parece lo que hemos aprendido ahora? Ya sabemos cómo tratar correctamente nuestros errores en nuestra aplicación. Ahora, en los siguientes cursos vamos a seguir trabajando sobre lo que aprendimos en este proyecto, con algunos tópicos ya más avanzados en lo que es seguridad en sí. Nos vemos.